miniSlicer 本地化流水线（用户指南 + 白皮书）

miniSlicer i18n 团队

May 11, 2025

Abstract

本文旨在给**最终使用者**（而非脚本开发者）一个一站式说明：为什么需要本地化流水线、能解决什么痛点、怎样一步步使用，以及如何按需扩展到新的替换场景。同时也为二次开发者附上架构细节与测试准则。

1 为什么需要这条流水线？

1.1 背景场景

- miniSlicer 及其扩展模块以英文界面为主，阻碍中文科研/教学场景推广。
- Qt Linguist 对大型脚本模块支持有限，且增量维护困难。
- LLM 出现后，批量高质量翻译成为可能，但缺少**安全**、**可回滚**、**可配置**的落地方案。

1.2 核心痛点

1. **文件量大**：数千个 `.ui` + 数万行源码，手工不可行。
2. **格式脆弱**：XML / C++ 任何格式错误都导致编译或运行崩溃。
3. **需求多变**：今天翻译 `<string>`，明天改模块 title，后天又要处理 `tr("...")`。
4. **缺乏增量**：UI 每次改动都需重跑，耗时且浪费 token。

2 方案概览

1. **可逆占位符**：以少见字符"¥"+10 位数字包夹，保证翻译文本定位唯一可还原。
2. **配置驱动（rules.yaml）**：把"要替换哪些字符串"写进 YAML；脚本通用，不再硬编码正则。
3. **四步流水线**：**标记** → **翻译** → **写回** → **清理与验证**。

4. **增量与测试**：基于 git diff 仅处理新增英文；pytest 单元测试验证规则。

图 2 给出整体数据流。

占位符 →LLM→sed→ 安全写回的流水线示意图

3 一分钟上手（Quick Start）

1. **准备环境**
   (a) Python 3.9+ 或 Nix；
   (b) 设置 OPENAI_API_KEY；
   (c) 可选：安装 xmllint 进行 XML 校验。
2. **克隆仓库 + 进入脚本目录** `git clone ... && cd script`
3. **运行命令**

```
make translate          # 一键翻译 UI + 源码（根据 rules.yaml
    ）
make fix               # 兜底清除 ¥XXXXXXXXXX¥ 占位符
```

4. **查看效果**：启动 miniSlicer；若仍有英文，可查看 tmp_*/report.log 确认规则是否遗漏。

4 可配置规则（rules.yaml）详解

4.1 YAML 结构

```
- name: qt_ui_string           # 规则名称
  ext: [ui]                    # 文件后缀
  mode: xml_xpath              # 抽取方式
  xpath: .//string             # XPath 表达式

- name: python_title
  ext: [py]
  mode: regex
  pattern: "(?P<prefix>\\.parent\\.title\\s*=\\s*(?:_?\\(\\s*)
     ?[\'\"])(?P<text>[^\'\"]+)(?P<suffix>[\'\"])"

- name: cxx_tr
  ext: [cpp,cxx,h,hpp]
  mode: regex
  pattern: "(?P<prefix>(?:::\\w+)?tr\\(\\s*[\'\"])(?P<text
     >[^\'\"]+)(?P<suffix>[\'\"])"
```

4.2 自定义规则流程

1. 复制 `rules.yaml`，修改或新增条目；
2. 可运行 `make dry_run RULES=my.yaml` 查看命中统计；
3. 调整无误后再执行 `make translate RULES=my.yaml`。

可在 `rules.yaml` 内同时声明 UI XPath 与源码正则，流水线一次性完成两类翻译，无需区分"UI/源代码" 子命令。

5 流水线分步解析

5.1 Step 1 生成占位符

脚本 `generate_placeholders.py` 读取 `rules.yaml`，对每条规则：

- 按 **ext** 过滤文件；
- 根据 **mode** 调用 XML/XPath 或正则；
- 对需要翻译的文本加前缀 `¥0000001234¥` 并复制到临时目录；
- 统计命中数写入 report，方便快速 review。

5.2 Step 2 调用 LLM 翻译

- 将所有带占位符行拼成 `all_strings.txt`，按 `CHUNK_SIZE` 分块；
- 使用统一 Prompt（可在 `translate_prompt.txt` 自定义风格）；
- 输出 sed 单行命令列表。

5.3 Step 3 安全写回

- 解析 sed → id: 中文 映射；
- 对 XML 用 DOM，源码用正则，回写中文并保持格式；
- 使用 `xmllint / black / clang-format`（若存在）二次校验。

5.4 Step 4 清理与验证

运行 `fix_placeholders.py` 根据规则定义的清理策略，确保仓库无遗留 `¥数字¥`。若仍检测到 → 脚本报错退出 CI。

6 常见问题 & 解决方案

症状 / 日志片段	可能原因与处理
`Invalid API key`	环境变量 `OPENAI_API_KEY` 未设置或过期。
XML ParseError	翻译文本含未转义字符 `&`；执行 `make fix` 重新清理并验证。
regex 未命中	在 rules.yaml 新增/修正 `pattern`。
CI 报"impure-derivations"	记得在 Garnix/Nix 加上 `--extra-experimental-features impure-derivations`。

7 进阶扩展

1. **多语言输出**：规则可加 `lang: ja`，脚本多次调用 LLM 生成多语目录。
2. **术语表驱动**：Prompt 中注入术语表，保证专业词一致。
3. **CI Bot**：失败自动在 PR 留评论列出未翻译行，方便 reviewers。
4. **可视化报告**：生成 `html` 展示规则命中率、示例 diff。

8 内部实现（给开发者）

- 脚本目录结构与调用关系图；
- 依赖列表（仅标准库 + 可选 `requests, tqdm` 等）；
- 单元测试：`tests/test_rules.py` 用示例文件断言标记/清理对等；
- 性能：对 10k 文件标记 <30s，写回 O(文件数)。

9 贡献指南

1. 提交新规则：需附 `examples/` 用例 + 单测；
2. 代码遵循 `black 23.7`；
3. PR 模板见 `.github/`；

10 许可证与免责声明

- miniSlicer 基于 APL 2.0；脚本遵循 MIT。

- 翻译内容由 OpenAI 生成，作者不保证绝对准确性，使用前请专业复核。